Onboard Flight Dynamic Route Optimization

João Portugal
joao.portugal@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

Janeiro 2021

Abstract

Nowadays the flight management systems (FMS) of an airplane can fly it autonomously from takeoff to landing with little intervention from the pilots assuming there are no anomalous events. However, some events require a diversion and subsequent route replanning. This replan is non-trivial, subject to a restricted criteria and takes attention away from the pilots and from air traffic controllers, attention that could be spent monitoring other flight systems. Route planning involves finding a new efficient route, communicate it to ATC, receive approval and reprogram the flight computer to follow the new path. Our work presents a module capable of finding a feasible flight path while complying with all the existing restrictions whether they are space restrictions such as weather events or no fly zones, or the airplane movement restrictions, removing the need of pilot intervention in this area lightening up their work as well as the air controller's work. We model the problem as an extension to the knapsack model and we then utilize a modified labeling algorithm created to solve the bicriteria 0-1 knapsack problem and efficiently recalculate the routes while taking into account all existing restrictions. The evaluation made was focused on both the accuracy of the route calculated and the time it took to recalculate the route and present the final result.

Keywords: Multi-objective problem, route planning, FMS, knapsack model, Flight routes

1. Introduction

Aviation is growing every year and is expected to double over the next 20 years [6] and more and more challenges surrounding airplanes such as safety, eco-friendliness and increasing air traffic become more and more discussed.

Airplanes are capable of flying autonomously from take-off to landing when considering standard normal operations but when trouble arises, pilot intervention is often needed [1]. Some of these issues require the pilots to adjust the route, sometimes the adjustment ending in a completely new destination.

The issues can be categorized into two different types: internal and external. Internal issues are problems the airplane might be having internally such as engine failure or a medical emergency. External issues are problems outside of the airplane such as weather storms or heavy traffic that might pose danger to the aircraft.

Some of the possible routes are already predetermined by the pilots prior to departure but sometimes a deviation is needed and when that happens the new route found may not be optimal. When considering what is optimal, there is a plethora of possible restrictions and criteria to follow but aircraft safety, fuel efficiency are usually the ones that matter the most.

Flight planning is the process of producing a flight plan that describes a trajectory to be taken from a start point to an end point. It involves calculating how much fuel is needed, the route that takes us to the arrival point safely while complying with ATC regulations and safety rules. Flight planning depends on a lot of factors. Depends on the distance, the weather conditions and the aircraft used. Companies are always required to take a surplus of fuel for safety reasons. This planning is not trivial and is never a one-time process. Varying weather conditions such as tail wind might increase or decrease the fuel needed. Weather storms might block some waypoints requiring the plane to take one another. It is important to note that an airplane has restrictions on its own movement, subject to its own flight dynamic restrictions, meaning that for example it can not change altitude instantaneously or change its course suddenly without affecting either passenger comfort or even the safety of the aircraft itself. This also has to be taken into account in flight planning.

There are already several flight planners able of generating routes, however when the aircraft is already airborne, it becomes dependent on the pilots manually adjusting the course of the route through

ATC advice. And this is something that can be theoretically fixed with an efficient on-board flight-planner that can receive input to changing conditions that might require route adjustments and automatically adjust it without taking away attention from the pilots, letting them monitor flight systems more closely.

We propose a flight-planner that can efficiently recalculate a route in the presence of adversarial conditions and automatically adjust the trajectory of the route while maintaining optimality, safety and being compliant with the restrictions imposed.

The rest of this chapter defines the objectives of this thesis,the solution proposed, and the outline of the thesis.

2. State of the Art

Pathfinding has been a problem studied throughout the years. It can be considered as the process of finding the shortest path between two nodes with or without obstacles in between. Djikstra's algorithm invented in 1956 [5] is the simplest form of pathfinding in a weighted graph. Since then, a multitude of pathfinding algorithms has been invented and improved on: A* [11] is an improvement on Dijkstra's algorithm where instead of just using the real costs per node, the search is now conduted based on heuristics. Ford-Fulkerson [8] is an algorithm that computes the maximum flow of a flow network, where we can find optimal paths in arcs that can have at max x flow. All this research has led to each situation requiring a specific algorithm and this algorithm could be adapted into different situations. For example, Bellman-Ford [8] can be used with or without a node queue and could be parallelized if some conditions are verified.

2.1. Pathfinding in transportation

In transportation, pathfinding has always been a problem. Everyone wants to get to their destination in the best way possible, but sometimes what is best is subjective. While some prefer to pay tolls to save time, others would rather go through toll-free roads and save money instead of time.

In 1931, the Zermelo's Navigation Problem [12] was the first posed optimal control problem. The problem consisted in finding an optimal path for a boat stuck in water with water currents and wind. If we do not consider the presence of these obstacles, the optimal path is a straight line from start to finish relating to the shortest path. However, when considering these obstacles, the optimal path is usually not a straight line. The same analogy can be used in aviation, obstacles can represent weather conditions or restricted airspace for example, leading to a path that is not a direct line from starting point to end point. Route planning has always been target of research and so there an uncountable number of thesis and works that describe multiple ways of finding the optimal path. We will describe some that pertains to our work the most.

The main goal of guidance applied to navigation is to provide a reference velocity, a path angle and a heading to enable the aircraft to follow waypoints $P_0, P_1, ..., P_N$ (Fig). However, there are multiple ways of finding these parameters. The FMS is already programmed so that if provided a waypoint, it can automatically give a path angle and heading into that location. The 3D path planning problem has been shown to be NP hard [3] but many solutions have been already presented.

In 2003 Myungsoo Jun et al. proposed a method of path planning using a map of thread probabilities made using surveillance data and from there create the optimal route [9]. It starts by determining occupancy values based on sensor readings and then applying the conditional probability of occupancy using Bayes' rule. Then, they generate a digraph from the probability map in order to convert the model to a shortest path problem. To find the optimal path, the authors chose to use Bellman-Ford algorithm [2] because of the flexibility it provides, since changes in the probability map are probable and would change the route and so they could update link lengths without having to stop and restart the algorithm. They also were able to make the Bellman-Ford asynchronous and distributed further improving efficiency. The results showed that the algorithm could generate paths although sometimes it was the safest path and not the optimal one. The main advantage of this solution is that it is very fast in computing a path and is compatible with distributed computation. However, things like frequent acceleration and deacceleration were not considered, which would need to be improved on, since this changes fuel consumption potentially leading to a sub-optimal path in the end.

In 2006 Igor Alonso-Portillo et al. proposed an adaptive trajectory planner capable of adjusting its world model and re-computing feasible flight trajectories in response to adversarial changes [1]. The module proposed was to be included into the FMS and it could transmit information that is not currently being transmitted such as engine failure, control surface jams, anything that could cause flight dynamics to vary and therefore modify the aircraft perfomance. The proposed module was not to be seen as a replacement of the FMS but as improvement to the system robustness. The module when detecting a failure updates the flight dynamic model accordingly, generates a footprint and starts a search for a suitable landing site. This landing site is chosen from an existing database of airports that contains information such as airport location, runway specifications amongst other relevant information. The footprint generated allows to find reachable airports. After this, the module performs a constraint analysis to select minimally safe airports and the constraints are repeatedly relaxed until at least one solution is found. Then an utility function is applied based on the airport characteristics to find the most suitable airport for an emergency landing. Finally, a trajectory to that airport is created utilizing existing tools on the FMS. Results showed that the module provided robustness to the FMS to different failure modes although more work on the generation of the trajectory was needed including taking into account wind and current weather.

In 2013, Nourelhouda Dougui et al. proposed the light propagation algorithm, an algorithm capable of generating sets of conflict-free 4D trajectories [6]. The algorithm is based on Fermat's principle of least action: *The path of a light ray connecting two points is the one for which the time of transit, not the length, is a minimum.* It finds an optimal path by computing smooth geodesic trajectories in environments with obstacles. Light tends to travel in low index areas where light rays are slowed down. Having this in mind, the algorithm programs obstacles as high-index areas, thus mimicking the light propagation behaviour. They were able to successfully apply on three different air traffic management problems. However, the authors noted that the algorithm still needed and could be optimized by computing the cluster resolutions in parallel and that the algorithm needs further improvements in more general restrictions, since it only accounted for temporal congestion and moving weather.

2.2. Knapsack Problem

2.2.1 Introduction

The knapsack problem is a combinatorial optimization problem and it talks about the common problem of packing the most valuable things without overloading the luggage. Formally, it seeks to select from a finite set of items, the subset that maximizes the linear function of the items chosen, subject to constraints [4].

A knapsack model can modeled into a pathfinding problem [4] thus an optimal trajectory can be found using this model. Indeed, we can choose as objectives what we want to maximize or minimize in our set, such as fuel costs or travel time and set the restrictions as to things like ETOPS rules, ATC rules, passenger comfort, ..., etc.

2.2.2 Solutions

In 2003, M. Eugenia Captivo et al. proposed a method in *Solving bi-criteria 0–1 knapsack problems using a labeling algorithm* [4] where we can formulate the knapsack problem as a shortest path problem [10] by transforming the multiple criteria knapsack problem into a multiple criteria shortest path problem over an acyclic network and then proceed to label it. It first converts a knapsack model to an acyclic network model.

The network generated by this algorithm has no cycles, every feasible solution of the knapsack problem has a corresponding path in the network generated from starting node s to end node t and the shortest path from s to t represents the optimal solution for the knapsack model and the value is the negative cost of the shortest path in the network.

This transformation is only for a single criterion but the generalization is very easy to do. Instead of just assigning one cost per arc, we assign a vector of costs per arc, each cost corresponding to one different criterion.

After converting it, to label it, they order it lexicographically but adding an additional property that other labeling algorithms do not have: the values concerning the first criterion are placed in non-decreasing order, while the values of the second one are placed in non-increasing order. So, to see if a new given label is dominated or not, it is only necessary to compare this new label with the last non-dominated label determined. For more than two criteria the new label must be compared with all the labels already determined.

3. Solution Proposal

All the previous work, while achieving relevant results to their own studies, none presented a clear module that worked efficiently that could be used online in an autonomous non-tripulated vehicle considering a wide number of restrictions. While some proposed a module for an emergency landing [7], it would be limited to certain types of aircraft (CS23). Others, do not consider wind variations or another type of restrictions in their planning.

Overall, the solutions proposed were designed for efficacy and not efficiency. Even if some factors such as wind or other factors were not considered in the solution, the main problem resides in that it would take too long for the current proposed flight-planners to find a feasible path for an on-going flight. The objective of this work would be to correct this issue that most flight-planners have while adding the dynamism that most flight-planners lack.

Our module receives as input the current trajectory in the form of: current position, optionally the waypoints that must be followed after and the destination point. These positions are represented in the form of *Latitude Longitude Altitude*. In addition to the trajectory, we receive a series of restrictions as well.

These restrictions are split in two big groups: Aircraft restrictions and route restrictions. Aircraft restrictions are constraints that the aircraft has such as speed, turns and fuel consumption. These restrictions are variable and depend on which aircraft model is being used. The route restrictions can be divided into two sets: dynamic and static. The static restrictions are set restrictions by the ATC known prior to departure to the pilots such as restricted airspace, also known as no-fly zones, or zones with limited speeds. Dynamic restrictions are restrictions that occur when the aircraft is already airborne. Weather storms, medical emergencies, all of these constitute dynamic restrictions.

As output our module returns the replanned route in the form of a start point, waypoints to be followed and the destination point. Each waypoint is represented as in the input.

In terms of space search, we decided that a straight line could be followed from point A to point B as long as the segment line formed by these points didn't break any of the restrictions imposed.

We reached the conclusion to use Captivo et al. labeling algorithm [4]. This algorithm is simple to use and is by itself very time efficient despite potentially using a large portion of space and ready to make updates in real time if adversarial changes happen.

Our module will only act if conditions change, meaning that only if we receive a signal that a condition has changed, only then we would verify the current route and changed it if needed be.

3.1. System Overview

Our system can be subdivided into 3 big phases: the input phase, the algorithm phase and the output generation where we run a backtracking technique to find the waypoints that lead to the solution found by the labeling algorithm.

The input phase consists in how we process the data that we will need later on to run the algorithm, initializing the data structures where we store the data, calculating the first part of the objective function and where we process the restriction set.

In the algorithm phase, we run all the necessary data through the algorithm in order to search for a feasible and optimal solution that complies with all the restrictions given.

The output generation consists in picking one solution from the optimal set and running a backtracking technique to find the set of waypoints that lead to the labels obtained by the labeling algorithm, these waypoints being the solution to the problem and then returning them.

3.2. Input

The input starts by receiving a starting point, then a number that represents the number of optional waypoints to be passed, the corresponding number of waypoints, and the destination point. This destination point is saved as the last member of the list of all waypoints that could be chosen in the solution path;

After, we receive another number that contains the number of restrictions we will receive, the restrictions and in the end, a few aircraft parameters such as speed, fuel and weight.

The restrictions are set by a type and then the corresponding parameters. A type 0 restriction expects a sphere, so the 4 following values will be the center of the sphere in form of latitude longitude, altitude and then its radius in kilometers.

The set of all existing waypoints is expected to exist prior to run-time and is loaded when the algorithm begins.

3.3. Objective function estimation

For this problem, we decided that having two objective-functions was enough to reach optimal solutions. The first one is to minimize the time needed to reach our destination and the second one is to minimize fuel consumption.

In order to find the first objective-function, we defined a line segment between our starting point and our destination point and the distance between the selected waypoint and the line segment - which forms a perpendicular line to the initial segment - is used to calculate the value. The farther the distance of the waypoint to the line segment, the lower the value as shown in Figure 4.1, where s and t are our start and end points respectively and A and B are waypoints. Therefore, the value of B will be higher because it is closer to the line segment than A.

To calculate the second-objective function, we could not immediately calculate the value of a waypoint because it depends on where the plane comes from. Figure 4.3 exemplifies this problem. We can not set the value of waypoint A or waypoint B because it depends on where it comes from and where it goes to. For example, if waypoint A goes to waypoint B it requires one amount of fuel but if it goes to waypoint C it requires a different amount. We then decided to leave this determination to when we run the labeling algorithm.

3.4. Restrictions

For the restrictions, there are two types of restrictions: the space restrictions and the airplane restrictions.

Space restrictions were defined as spheres that could be placed anywhere.

They have a radius and a center with coordinates in latitude, longitude and altitude. It becomes easy then to calculate if the arc formed by the waypoints intersects with a sphere. Figure 4.4 represents a no-fly zone issued around the house of at the time president-elect of the USA, Joe Biden, 7th November 2020.

But, in terms of relating these restrictions in function of the waypoints, to put them as weight restrictions in the knapsack model, it was impossible to do that because for each new restriction added as a knapsack weight restriction, the labeling algorithm would need to be adapted, so that it ensured all the weights were being respected and also because the weight of each waypoint depended on where it came from. So we decided to create only one restriction function that would serve as a weight limit to the functions and the rest to be accounted for in run-time of the labeling algorithm.

The restriction we decided to use in the knapsack model was the maximum of fuel reserves that the airplane had baring 30 minutes of fuel and the problem here was the same to the problem we faced in the objective function calculation where we couldn't calculate the fuel consumption solely based on a waypoint, only on a set of waypoints. And so what we ensured was that the sum of all waypoints considered could not be higher than the fuel capacity of the airplane defined above.

3.5. Data Structures

Data structures are particularly important in our problem since we require that operations need to be as fast as possible. This includes finding $S(j^a)$, the restrictions set and the waypoints in an efficient way. To do this, we chose different implementations for each type, that would fit best or at least not compromise the running time of the algorithm.

3.5.1 Waypoints

Since waypoints are always directly accessed, we have them stored in an array. Before running the algorithm, we run a merge-sort algorithm to order them based on distance to the starting point. In all three cases (worst,average,best), merge-sort has a time complexity of $\mathcal{O}(n \log n)$. Since we access directly the waypoints, then the time complexity here is simply $\mathcal{O}(1)$ for each access, and in the end, since we will consider all the waypoints for the solution we have the final time complexity of $\mathcal{O}(n)$.

3.5.2 Data structure S

We denominate S as the data structure represented in the labeling algorithm that will always be accessed through indexes. We always know its key, for example in $S(j^a)$ the key will be j^a, which is unique for each node. So, we decided to model S as an hash table. Collisions in this hashtable are solved using linear probing. If the number of elements present in the hashtable is half of the maximum number of elements allowed in the hash table, we rehash the old table into a new one with double the size. This further prevents collisions. The amortized cost of inserting an element into an hashtable, even considering the eventual rehash is $\mathcal{O}(1)$.

3.5.3 Restrictions

The restrictions are saved using a linked list and for each arc AB, we see if it breaks any of the restrictions stored in the linked list. Since we have to verify them all and considering we have r restrictions then the cost of this operation will be $\mathcal{O}(r)$. Although we do not expect a high number of restrictions, this is something we have to take in mind in testing and if needed further improving since it could bottleneck our module.

4. Labeling Algorithm

In this chapter, we explain how the base labeling algorithm works, what modifications we made to making it capable of handling our problem and what optimizations we made in order to increase its efficiency and efficacy. Section 5.1 describes the base labeling algorithm, section 5.2 the modifications we made to the algorithm. Section 5.3 makes a summary of this chapter.

4.1. The base algorithm

The base algorithm, proposed by Captivo et al. [4] intends to maximize N objective functions subject to one restriction with a maximum capacity.

It starts by initializing S, where $S(j^a)$ is the set of non-dominated labels concerning the set of all paths from s to j^a, setting $S(1^0)$ with label $(0, 0, ..., 0)$ and $S(1^{w_1}$ with label $(-v_1^1, ..., -v_r^1)$ where r is the number of items. It also creates two lists, T and V, where V starts out empty and T with two values, 0 and w_1, the weight value of item 1. These two sets contain all the labels of layer 1.

Then, for the remaining layers j $(j = 2, ..., n)$, the set $S(j^a)$, where $a \in 0, ..., W$ is built as the following:

1. If j^a has only one arc incoming then consider two different cases:

    (a) if the arc $((j-1)^a, j^a)$ belong to T but $a - w_j$ does not, then $S(j^a) = S((j-1)^a)$

(b) if the arc $((j-1)^{a-w_j} j^a$ exists, i.e a does not belong to T but $a - w_j$ does then labels in $S(j^a)$ can be defined by summing $(-v_j^1, ..., -v_j^r)$ and $S(j-1)^{a-w_j}$

2. If two incoming arcs, a and $a - w_j$ then the labels in $S(j^a)$ by choosing the non-dominated labels simultaneously from the labels in $S((j-1)^a)$, and from the labels obtained by summing $(-v_j^1, ..., -v_j^r)$ to each of the labels in set $S((j-1))^{a-w_j}$

The problem solutions, $S(t)$ can be given as the set of non dominated labels of $\bigcup_{a=0}^{W} S(n^a)$ and the items chosen can be found by using a backtracking technique.

Since we are in a multi-objective problem, there can be more than one solution to pick-up from. We arbitrarily decided to choose the one that minimizes flight-time.

Backtracking is an algorithmic technique to solve a problem by recursively try to build a solution incrementally, one piece at a time, removing pieces that fail to satisfy the constraints given at any point in time. In this case, we have two objective functions and so we could backtrace both functions to find the items that satisfy both conditions. Since recursive calls are computionally expensive, what we do is to find all solutions that satisfy the first objective function, save the indexes of the items that lead to the solution and then run a loop where we try to satisfy the second objective by calculating $f_2(y)$ with the items we considered.

4.2. Modifications to the base algorithm

The base algorithm was made with a very specific problem in mind, the knapsack problem. However, since our problem is not directly related to knapsack, we need to adapt it in order to function with our problem.

The algorithm is made considering only one single capacity restriction and not multiple restrictions that could hinder movement, like no fly zones, but that are in no way related to the item set and still affect the order in which items are selected. We thus decided that the capacity restriction, W, should be the fuel the airplane was carrying, leaving the rest of the restrictions to run-time verification.

Since we have more than one restriction in our problem, we had to change the algorithm.

We start by running merge-sort on the waypoints, ordering them by distance to starting point S.

As referenced in chapter 4, it's impossible to estimate a weight cost of an item, solely based on it. It has to take into account where it comes from. We then added the change, that every time we change the value being assessed a, we recalculate all the values that w_j can become because we know where it comes from if $S(j-1)^a$ is defined. We add a last item parameter to every $S(j)^a$, referencing the last item added to the set at that point. For example, if j is added to the possible solution set, then the last item parameter of $S(j)^a$ is j. If j is not added to the solution set but $S(j)^a$ exists, then the last item in $S(j)^a$ is equal to the last item contained in $S(j-1)^a$. The last item in $S(1^0)$ is the starting point s and the last item in $S(1^{w_1})$ is 1.

Since the weight of an item can then assume multiple values,we calculate all those values and introduce a new third loop, where we iterate through all the possible weights of this item.

Every time we consider adding an item to the solution set, we have to consider all the existing restrictions and if they're broken if we add this item. Since the restrictions rely on knowing where the plane is coming from, we fetch the last item where the plane was headed from $S(j-1)^a$.

The first situation in the algorithm where we create $S(j^a)$ is if only $a \in T$. We then verify if the restrictions are broke from $S(j-1)^a$ to $S(j^a)$. If not, we copy the labels from $S(j-1)^a$ to $S(j^a)$.

The second situation is if $a - w_j \in T$ and $a \notin T$. Then, we have to verify if going from the last item in $S(j-1)^{a-w_j}$ to j results in breaking a restriction. If it does, we discard the arc and nothing is created. If not, we add the labels in $S(j-1)^{a-w_j}$ plus the values of flying from the last item to that item.

The third situation is if both $a - w_j \in T$ and $a \in T$. We separately verify if j is achievable by both $S(j-1)^a$ and $S(j-1)^{a-w_j}$ Then we proceed as follows:

1. If only $S(j-1)^a$ complies with the restriction set, we proceed as if we were in the first situation.

2. If only $S(j-1)^{a-w_j}$ complies with the restriction set, we proceed as if we were in the second situation.

3. If both cases comply with the restriction set, we then choose the non-dominated labels from both $S(j-1)^a$ and $S(j-1)^{a-w_j}$ plus the sum of the values of flying from the last item to the new item.

4. If no case complies with the restriction set, we discard the arc and continue.

For simplicity we consider that only sphere restrictions are possible in the set of space of restrictions, and since we have two waypoints (origin and destination) we can calculate if the arc composed by those two points intersects any of the existing space restrictions.

To do this, we introduce a change in the base algorithm, where if the condition $a - w_j$ was verified, meaning that an arc where the item was included, we first confirm that this item can be added to the item set considering where it came from $S(j-1)^a$.

4.2.1 Restriction verification

When running the algorithm, restrictions need to be verified in case a trajectory is not possible. The plane does not travel through a flat surface, it travels in arc around Earth, which can be either considered an ellipsoid or a sphere. For this work, we considered it a sphere, though it is trivial to convert the calculations into an ellipsoid which is adding a second radius.

To see if the arc that forms the path from A to B, and considering C as the center of the sphere that represents the restriction we do the following:

1. convert A, B, and C coordinates to Cartesian coordinates. Note that, when converting to Cartesian coordinates, R will be the radius of the earth and we will have to sum the altitude of the waypoint considered.
2. A, B and O, where O is the center of the Earth define a plane that contains arc AB. We calculate the normal vector normalized to this plane, that is given by $(A \times B)/N(c)$ where $\times$ represents the cross product and N(c) is the norm of the cross product between A and B.
3. With the norm, we can now calculate the distance from C to the plane given by $C \cdot n$ where $\cdot$ represents the dot product.
4. Now we can compare the distance obtained with the radius of the sphere. If this distance is smaller or equal to the radius, then the arc intersects the sphere. If the distance is bigger, then the arc does not intersect the sphere.

For the aircraft restrictions, a very simple model was created in which we set the max bank angle as 30º, which will affect the turn time, and the maximum vertical rate of the plane was 3000 feet per minute. This is trivial to calculate, all we have to do is to see if the time it will take to change altitude at the maximum rate surpasses the time it will take to travel from point to point and if it does then the airplane would need a higher rate which is not possible.

4.3. Output

The labeling algorithm outputs a set of solutions which is a Pareto front, or none if no path to t is feasible. Considering that a set of solutions existed, we needed to pick a solution from this set. We arbitrarily chose to pick the one that minimized time spent.

From there, we can then run a backtracking technique to find all the solutions for objective-function 1. The backtracking technique consists in a dynamic-based approach where we check if it's possible to reach the solution from the current element. Fundamentally, this consists in the well-known subset-sum problem in which the target is the first-objective and the set, the first objective function. The dynamic approach to the subset sum problem is made over a integer set. We multiplied our value function by 10 and discarded the rest of the decimal part to convert it from a real set to a integer set.

If a solution set is found, it verifies if that set of waypoints chosen complies with objective-function 2 and if it does, we have found the solution. If not, it continues to exhaust every option until it finds one.

After all this, we can now produce the final output. This final output consists in the list of waypoints that forms the optimal path from s to t in the form of latitude longitude altitude, that begins with the starting point, the waypoints to be followed and the destination point.

The dynamic programming implementation of the backtracking technique has a $\mathcal{O}(n * target)$ exploration time complexity and up to $\mathcal{O}(2^n)$ time complexity for the recursive calls to find the solution set.

5. Experimental evaluation

In this chapter, we define what we are looking for when testing the module, the setting where we tested and how our tests were generated. We also present and discuss the results obtained.

5.1. Setting and objectives

For this study, we consider that the space was restricted to the Iberian Peninsula and that the altitudes could vary randomly between 0 feet and 30000 feet.

The experiments were led on a i5-8400 running at 3.8GHz with 8GB RAM available. The whole module was written in C language (C11) and both the input and the waypoint database were randomly created using a python script using the random.py library.

The objectives for these experiments were to find if the algorithm could find solutions in efficient time when varying the number of waypoints, the number of restrictions and the capacity, these three factors being the main sources of time consumption in the labeling algorithm. Memory usage by the labeling algorithm was also measured.

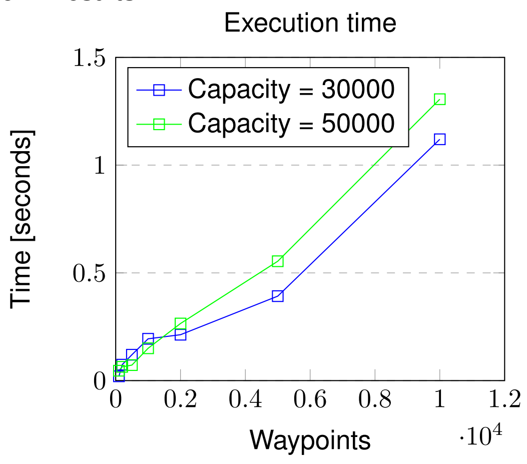
The restrictions were generated using a maximum altitude change rate of 3000 feet and with each circle having a random number between 0.5

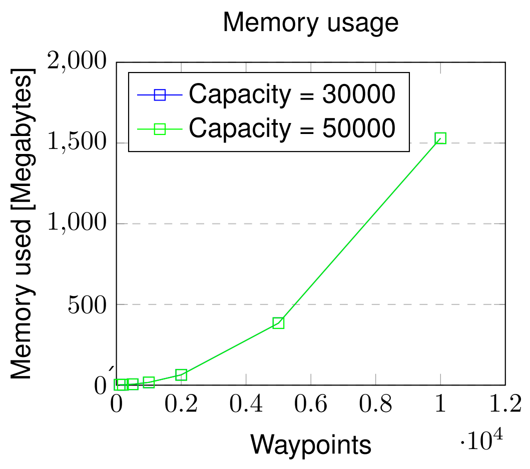
WP	Res	Cap	Time	Mem
100	30000	100	0.020	2.5
200	30000	100	0.074	2.5
500	30000	100	0.120	5.7
1000	30000	100	0.194	17.4
2000	30000	100	0.213	64.2
5000	30000	100	0.392	384.1
10000	30000	100	1.120	1529.6
100	30000	500	0.014	2.4
200	30000	500	0.017	2.9
500	30000	500	0.032	5.6
1000	30000	500	0.058	17.5
2000	30000	500	0.112	63.3
5000	30000	500	0.385	384.4
10000	30000	500	1.013	1529.7
100	30000	1000	0.012	2.5
200	30000	1000	0.017	2.5
500	30000	1000	0.033	5.7
1000	30000	1000	0.055	16.8
2000	30000	1000	0.115	63.4
5000	30000	1000	0.381	384.5
10000	30000	1000	1.035	1529.5
100	50000	100	0.046	2.6
200	50000	100	0.064	2.6
500	50000	100	0.072	5.6
1000	50000	100	0.150	17.6
2000	50000	100	0.265	63.4
5000	50000	100	0.554	384.1
10000	50000	100	1.306	1530.1
100	50000	500	0.025	2.4
200	50000	500	0.028	2.9
500	50000	500	0.042	5.6
1000	50000	500	0.103	17.4
2000	50000	500	0.184	63.4
5000	50000	500	0.520	384.6
10000	50000	500	1.286	1529.3
100	50000	1000	0.025	2.6
200	50000	1000	0.040	2.7
500	50000	1000	0.055	5.6
1000	50000	1000	0.098	17.2
2000	50000	1000	0.193	63.4
5000	50000	1000	0.504	384.3
10000	50000	1000	1.310	1529.7

Table 1: Execution time of the labeling algorithm.

and 5 kilometers.

5.2. Results





5.3. Discussion

5.3.1 Labeling Algorithm

The labeling algorithm executes in a very fast time, yielding promising results even in the highest numbers tested. In terms of real time efficiency, 1 second for the labeling algorithm considering a plane at 400 knots, consists in approximately 200 meters, which is a very small change causing little to no change if the algorithm was reran at that point.

Despite, the number of restrictions rising from 100 to 1000, little change was verified. This is due to the fact that, while in the worst case scenario the time could go substantially up, in average, the restrictions limit the number of labels being created, thus severely limiting the third loop.

The capacities tested can be considered for low flight fuel in the form of 5000, a flight fuel that could be considered for a plane to be in mid-flight with 30000 fuel left and a flight fuel that could exist in the departure stage of the travel in 50000. Noticeably, the capacity is the most impactful factor since when it changes, it usually changes in the order of tens of thousands, from 50000 to 60000. If we consider a Boeing 747-400, it can take up to 216,840 litres of fuel, a big change from the 50000 maximum considered.

The number of waypoints influence the time as well despite not so much, the highest number, 10000, being a obscene number of waypoints, tested just to prove that the algorithm could handle such a number.

In terms of space, the algorithm requires a lot of space, especially if the number of restrictions are not suited. For example, if the restrictions are all set in America and the waypoints are all set in Europe, the algorithm will generate every single possible combination of waypoints granted the capacity allows it. So, if both conditions are true, we then have a $\mathcal{O}(\sum_{j=1}^{n} j!)$ of nodes, which is a insurmountable number. Since our tests were randomly generated, some of them were bound to fail, since this possibility existed.

In terms of efficacy, a small number of tests was made to ensure the results were right. But even so, no real input was introduced and compared with real cases. So, in the long range, some problems might arise that the solution is not optimal.

Further testing should be made using real world cases, where waypoints have a very limited number of waypoints they can go to, the other part being limited by the restriction set, making the space limit as low as possible. If the number of restrictions is low and the restriction dimension is low so that it covers a low area, a lower number of waypoints should be considered to ensure that the algorithm does not run out of memory.

5.3.2 Backtracking technique

For the backtracking technique, up to 5000 waypoints were tested. The results were good, in the highest number of waypoints, the total time was 11.164, which considering a plane at 400 knots it is about 2.2 kilometers.

For a short to mid-haul flight, a very low number of waypoints in 2000 can be considered, making the time around the 5 second mark. The long haul can take more than 5000, although in the time it would take to calculate it, the conditions would change, so the alternative could be to search the solution space from the current point, to a point where pilots know they will cross.

In this algorithm, we intend to minimize the objective-functions and so, the sum the backtracking technique will be searching for, will be as low as possible, making it as efficient as it can be, both in terms of search time and search space. This problem is an on-going investigation and if found that the classical problem can be solved in polynomial time, that $\mathbf{P} = \mathbf{NP}$, then this part should be updated so that it can solve the problem faster.

6. Conclusion

Flight planning is becoming more and more demanding in terms of time efficiency since airspace is becoming more and more overcrowded. While the effects of the on-going COVID19 pandemic will certainly make its mark in the aviation world, aviation was expected to double over the next 20 years. Pilots need to rely less on air traffic tower and be able to quickly re-plan their route if an issue arises.

The current system for pilots to change the current flight path is to search where the adversarial condition is when informed by the air traffic controller and check which viable waypoints are closer to that point so that they can travel to that instead. After that, they still have to inform ATC about the new route they intend to take and await confirmation. This process is not trivial and takes a lot of time both from the pilots and from the air traffic controller, time that could be well spent doing other tasks.

We propose a flight planner that is efficient and effective in finding a route while avoiding restrictions, which can replace the process mentioned above. The algorithm used can work with any type of restrictions since it does not depend on them. It also can take any number of value functions making it desirable for companies that want to maximize the value of a route, to their own standards of value and spending. Whether they wish to minimize fuel consumption or flight time or passenger comfort, all they have to do is create a mathematical function for that function, and run the algorithm over it.

The algorithm used consisted in an adaptation from the labeling algorithm proposed in 2001 by Captivo et al. [4]. This base algorithm worked by modeling a knapsack model into a shortest path problem, by layering the items and creating two arcs between them if possible, one when the item was added into the solution set and one where the item was not in the final solution set.

The modifications made were designed so that multiple restrictions could be fitted and still respected, since the base algorithm only supported one restriction at a time. The algorithm verifies if all the restrictions are not broken, and if they are not, then the corresponding arc can be created. The main factors influencing the execution time are the number of waypoints, the current fuel of the airplane and the restriction set.

The labeling algorithm returns the set of non-dominated labels from the starting point to the destination point. In this work, we arbitrarily chose one, since they are all optimal. From there, a backtracking technique using dynamic programming was used to find the waypoints, the first value of the label corresponded to. Then, we calculated if the second value corresponding to the value in the second-objective function. If it did, the final solution was found. If not, the backtracking technique would continue until it found the solution set. The final output consisted in the starting point, waypoints to follow and the destination point.

The results of running tests over the labeling algorithm and the backtracking technique were good where the times were not big enough so that the distance flown by the airplane while the result was being generated created enough change for the result to be different if calculated again at that point in time. Results showed that it is possible for an on-board flight-planner to exist and dynamically adjust flight routes if initial conditions change.

7. Future Work

Despite good initial results, several improvements can be made and should be explored. Initially, it would be very beneficial to have feedback from flight-planning specialists. This would allow the validation of the current system in place, and further improvement could be made over it.

As mentioned before, the algorithm can take up to a lot of space, especially when the number of waypoints is high and the number of restrictions is low, since a very high number of labels will be created. A potential solution to this problem would be the longest common sub-sequence in which only states near the optimum would be created. This could reduce the number of labels created and massively improve the space usage of the algorithm.

The algorithm was setup by always having 2 objective functions. It would be interesting to explore the cost of having 3 or more objective functions which would lead to a maximization of profit by companies that want it. However, for more than two criteria, it is important that it is not enough to compare the new label with the last label added and instead compare it to all others added to the node, which is a very time consuming operation.

When choosing a solution from the solution set, in this work we arbitrarily did so. It would be a good change to explore ways of picking a solution, whether is using an algorithm that finds the knee point in the Pareto front or creating a framework where the user can express their preference for which objective to prioritize.

Although the results did not suggest the restrictions were causing a huge time factor in the system, the current method in place is not the most efficient one. Even if the waypoints being analyzed are not close to a space restriction, they can even be on one side of the map and the restriction on the other, the restriction will still be checked. Implementations of new data structures that can efficiently obtain which restrictions are close to the waypoints should be researched and explored.

Currently, despite the algorithm being able to receive any type of restrictions, the restrictions themselves are currently just the spheres and the altitude rate change . It would be interesting to modify those to more different types of, to really cement the model as a very adaptive module.

In this work, a very simple airplane model was used. Better flight dynamics should be considered and implemented, despite the labeling algorithm being independent from this. The cost-functions are simple and do not require a lot of operations. If a lot of trigonometry is needed, this could impact the execution time, so this should be taken into account

When obligatory waypoints exist, concurrent programming could improve the running time by running two or more algorithms concurrently between this waypoints to reach a faster solution.

Turn time and speed variation were not considered in this work which could change the solution in some cases, making the solution less optimal that what it should be. Frequent altitude changes were also not considered for the second objective which can greatly influence the fuel consumption.

Testing should be made using real world cases and real world restrictions. For this to happen, a rigorous flight dynamic model should be in place, to ensure the solution set found by the labeling algorithm is the right one. These tests will serve to confirm and ensure the efficacy of the algorithm.

References

[1] I. Alonso-Portillo and E. M. Atkins. Adaptive Trajectory Planning for Flight Management Systems. Technical report, University of Maryland, Maryland, 2006.

[2] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[3] J. Canny and J. Reif. New Lower Bound Techniques for Robot Motion Planning Problems. *Annual Symposium on Foundations of Computer Science (Proceedings)*, pages 49–60, 1987.

[4] M. E. Captivo, J. Clímaco, J. Figueira, E. Martins, and J. L. Santos. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Computers and Operations Research*, 30(12):1865–1886, 2003.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[6] N. Dougui, D. Delahaye, S. Puechmorel, and M. Mongeau. A light-propagation model for aircraft trajectory planning. *Journal of Global Optimization*, 56(3):873–895, 2013.

[7] A. Fallast and B. Messnarz. Automated trajectory generation and airport selection for an emergency landing procedure of a CS23 aircraft. *CEAS Aeronautical Journal*, 8(3):481–492, 2017.

[8] L. Ford and D. Fulkerson. Maximal Flow Through a Network.

[9] M. Jun and R. D'Andrea. Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments. pages 95–110, 2003.

[10] J. B. Orlin. *Ahuja, Magnanti, Orlin - Network flows Theory, algorithms and applictions*. 1993.

[11] B. R. Peter E. Hart, Nils J. Nilsson. A Formal Basis for the Heuristic Determination of Minimum Path Costs, 1968.

[12] E. Zermelo. *ber die Navigation in der Luft als Problem der Variationsrechnung*. 1930.